Image Steganography Removal

**Functional Specification**

|  |  |
| --- | --- |
| Author | Luke Parsons |
| Revision | 0.1 |
| Date | 18 June 2017 |

Table of Contents

[1 Overview 3](#_Toc478725794)

[2 Requirements 3](#_Toc478725795)

[3 Recognition 3](#_Toc478725796)

[3.1 Format Type 3](#_Toc478725797)

[3.2 ZFM: Create Handler 3](#_Toc478725798)

[3.3 Signatures 3](#_Toc478725799)

[3.3.1 Extra-Data field 3](#_Toc478725800)

[3.3.2 Document.xml 4](#_Toc478725801)

[3.3.3 Document.xml.zenc 4](#_Toc478725802)

[3.4 Functional visual aid 5](#_Toc478725803)

[4 Handling 5](#_Toc478725804)

[4.1 ZIP handler 5](#_Toc478725805)

[4.2 Handler creation 5](#_Toc478725806)

[4.3 Modifications 5](#_Toc478725807)

[4.4 Exposing child items 5](#_Toc478725808)

[4.5 Format Facets 5](#_Toc478725809)

[4.6 Licensing 5](#_Toc478725810)

[4.7 Configuration options 5](#_Toc478725811)

[4.8 Properties 5](#_Toc478725812)

[4.9 Responses 5](#_Toc478725813)

[5 Appendix A – MMAP\_recogniser.mmap 7](#_Toc478725814)

[6 Appendix B – Functionality Flow chart 8](#_Toc478725815)

# Overview

This document describes the functional requirements for an SDK which will focus on the destruction of steganography within various image formats supported by the OpenCV and ImageMagick libraries. Steganography in the context of this document is the practice of concealing potentially malicious data within an image, concerning itself more with hiding the fact that there is covert content than actually obfuscating the data itself like cryptography.

As this SDK will later be used to integrate Steganography destruction into the Clearswift DCI engine it must fulfil the following functionality set out in this document. While this document focuses primarily on the removal of steganography within images it’s important to make clear that images make up one of several vessel types that information can be hidden in. A vessel is the file or file type being used to transport the hidden information within.

There is potentially room for future extensions so this SDK will make up part of CML, sub-category Steganography Removal Library (**SRL**).

# Requirements

This solution must;

* Destroy Least Significant Bit (LSB) steganography in these image formats with no noticeable effect to the end user
  + And possibly other forms of steganography
* Be able to handle a large number of images
* Provide an interface to handle a wide range of image formats simply
  + Provide the building blocks so that the user may extend it themselves
  + Return the images to their original image format after the removal is complete
  + Be intuitive in its usage and well documented

# Destroy LSB Steganography

The way in which Least Significant Bit steganography works is by using the innate weakness of the human eye to perceive minor changes. Each pixel in an image represents and RGB value made up of 3 bytes for each of the primary colours. By changing the least significant bit the overall colour value of the pixel changes by at most 3 which in the RGB spectrum (made up of 2563 or 16,777,216 colours)is impossible to detect by the human eye.

The way in which this SDK destroys the information hidden within this colour differential is by using the **lossy** image compression algorithm JPEG (or JPG). By using a perceptual model based loosely on the human psychovisual system it removes the details we are unable to detect and further compress the image. By encoding the images in this way many of the bits making up the originally embedded bit-stream will be lost and on attempt to retrieve it will provide meaningless data.

## Possibly other forms of steganography (MAYBE NOT NECESSARY?)

While it is true this destroys LSB steganography it is possible that this also deals with other types of steganography that we aren’t aware of haven’t tested, purely because of the fact we are using a lossy algorithm to encode the images.

# Handling

## StegImg handler (INCOMPLETE)

The handling for all the images will be handled internally by the StegImg handler which will possibly derive from an existing image handler (if it exists). While it will be fairly simple in what it must do, instances where there are many images to sanitize there will be a lot of processing and image handling to deal with so optimizations should be made where possible.

## Modifications

The only modifications made to the image will be to encode it using JPEG and then return it to its original format.

## Licensing

No licensing control mechanism is to be implemented as this will be for the sole use in the Clearswift DCI engine.

## Configuration options (ASK ABOUT LOSSLESS COMPRESSION FOR 0\*)

This will support only one configuration option; image\_quality. This value will provide the user with a number of options for the quality to compression ratio. These values are fixed because there is a compression factor at which image size no longer decreases and the quality drastically reduces. These are;

* 100 = HIGH \***represents using the lossless JPEG2000 algorithm**
* 85 = STANDARD: Compression factor 15 which is the optimal factor at which LSB Steganography destruction is guaranteed and the loss in image quality remains unperceivable
* 70 = LOW compression factor 30 which provides a lower image size and therefore a higher chance of disturbing or destroying the Steganography contained within but reduces image quality further.

## Properties

This handler will populate the ***m\_processing\_status*** member of the ***Steg\_img*** class with a default value of UNCLEANED, and after processing will either be CLEANED, UNCLEANED\_ERR indicating that an error occurred during processing in which case the error value can be attained from the ***m\_errval*** member.

# SDK interface

## Building Blocks

As the SDK is intended to be integrated into the engine it is possible that functionality in the future may need to be extended in such a way that isn’t provided by this SDK. In that event the user should be given all the tools necessary in order to extend the functionality as required, including diagrams and documentation. These are specified further below;

* Source code for the library
* Visual Studio Project VS2013
* Linux Makefile
* Include files for SDK
* Doxygen/SDK Documentation on how to use the library
* Samples on how to use the SDK

## Return to original format

In order for this SDK to appear seamless in its interface it’s important that after encoding the image using JPEG that the image be returned to its original format. To the end user we must appear to be doing as little as possible to the image, and just changing the extension of the image is not enough. Both ImageMagick and OpenCV have image converting capabilities and between them should be capable of handling the vast majority of the common image formats passing through a gateway.

The exact list of image formats that these two libraries provide will be non-inclusive meaning instead of checking for image formats that are supported, it checks for whether it isn’t supported which can be updated as encountered.

## Intuitiveness and Documentation

For the SDK to be as easy to use as possible in the future it’s important that it remains intuitive and easy to manage resources with. For example, all allocation and de-allocation of objects will be enforced using a resource handle owned by the StegImg\_handler through calls to ***make***\_***X***() so as to reduce any potential data leakage in a performance critical system.

All of the functionality that the library provides, as well as samples on how to use it will be included through Doxygen documentation to facilitate the process. The UML diagrams encompassing the SDK and the systems within are included in the appendix as well as use cases where appropriate for clarity.